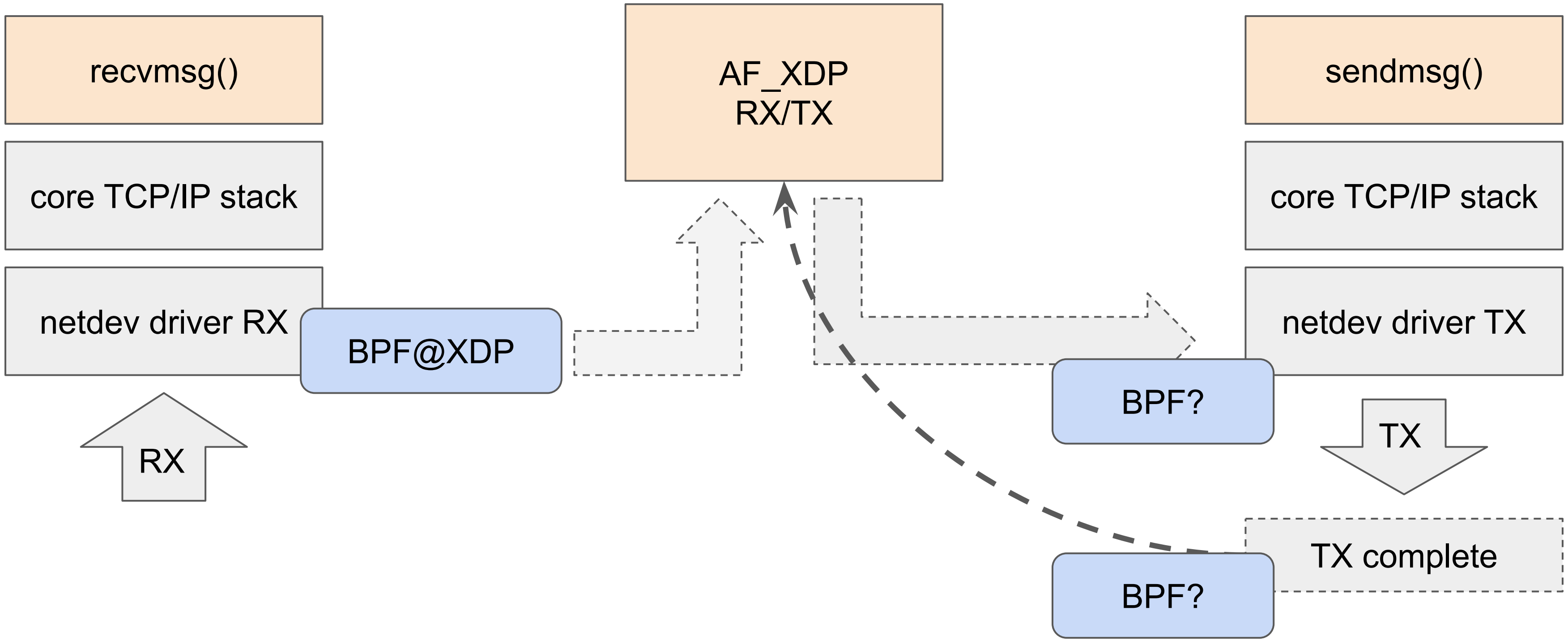
XDP Metadata

sdf@google.com
May 2023

XDP introduction



XDP metadata on RX

- NIC can parse and export a bunch of per-packet metadata
- Avoids kernel re-calculating the same info
- Examples
    - RX Flow Hash
    - RX Checksum value (CHECKSUM_COMPLETE) or status (CHECKSUM_UNNECESSARY)
    - RX Hardware timestamp

XDP metadata on TX

- Kernel can signal a bunch of things at TX for NIC to offload
- Avoids kernel wasting CPU on moving the data/etc
- Examples
  - Offload L4 checksum (i.e., ask nic to calculate checksum [from...to] and write at given offset)
  - GSO - split single big packet into MTU-sized chunks
  - Request HW TX timestamp
  - Tunneling/VLAN offloads (ask NIC to slap extra headers)
  - Pacing (SO_TXTIME / EDT)
  - Toke's XDP queueing
    - https://lwn.net/Articles/901046/
    - out of scope here, but something to keep in mind

Why do we need it?

- skb feature parity
- core tcp/ip stack already using all these offloads
- we need to expose them to XDP / AF_XDP context to be more CPU efficient
  - or access things that are otherwise inaccessible (like HW timestamps)

Where we are?

- RX side is implemented (the framework + small amount of kfuncs)
- Each metadata is exported via separate kfunc
  - bpf_xdp_metadata_rx_hash
  - bpf_xdp_metadata_rx_timestamp
- netdev-bound programs (to avoid netdev->kfunc indirect calls)
  - resolve kfunc at load time to direct calls
  - make sure this program can be attached only to "bound" netdev
  - make sure progs become orphaned (i.e. unusable) when netdev goes away

What's missing?

- more RX helpers (checksum is the obvious one)
    - I'm assuming whoever needs it can send out the patches
- TX part is completely missing

What are the use cases?

- From our POW, we just need:
  - RX HW timestamp (done)
  - TX HW timestamp
    - which means we need TX completion and access to TX completion descriptor
- But it doesn't seem fair to solve only completion part
- Having a TX metadata framework seems like a good way to scope it

What do we really need?

- Two hooks
  - XDP for every egress packet
  - XDP for every egress packet completion
  - is it too much?
- Access to real TX descriptors
  - from the kfucs
- Access to AF_XDP umem chunk?
  - We might want to put tx-timestamp into AF_XDP umem chunk for userspace consumption
- per-device kfuncs to get/set the metadata

Alternatives considered

- Something AF_XDP specific?
- Full-blown XDP@Egress?
- HID-BPF like hooks in the driver's egress path?
  - this is the one I've mostly settled on
  - device-bound tracing programs with TX metadata kfuncs

Proposal

- syscall program to attach to new hooks
- tx/tx_completion metadata kfuncs
- netdev-bound tracing programs (to resolve metadata kfuncs)
- AF_XDP can also use these for the offloads in the future, for example:
  - bpf_xdp_metadata_tx_l4_checksum_offload(l4_offset, TCP);

Example

- SEC("syscall")
  - bpf_xdp_attach_egress_prog(ctx->ifindex, ctx->egress_prog_fd);
  - bpf_xdp_attach_egress_compl_prog(ctx->ifindex, ctx->egress_compl_prog_fd);
- SEC("fmod_ret/xdp_egress")
  - bpf_xdp_metadata_tx_request_timestamp(ctx);
- SEC("fmod_ret/xdp_egress_compl")
  - bpf_xdp_metadata_tx_timestamp(ctx, &sample->timestamp);

Pros & Cons

- Pros
  - easy to experiment
  - not a UAPI
- Cons
  - two new hooks
  - driver specific
    - same as XDP though?
  - XDP-like, but not XDP
    - no access to AF_XDP metadata
    - no access to existing XDP helpers